

A RUST BASED, ECS PATTERNED, INTERACTIVE FLIGHT SIMULATOR

THESIS

Chad A. Willis, 2d Lt, USAF

AFIT-ENG-MS-21-M-XXX

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

###### AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.AFIT-ENG-MS-21-M-XXX

A RUST BASED, ECS PATTERNED, INTERATIVE FLIGHT SIMULATOR

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in xxxxx

Chad A. Willis, BS

2d Lt, USAF

March 2021

DISTRIBUTION STATEMENT A.  
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-21-M-XXX

A Rust based, ecs patterned, interactive flight simulator

THESIS

Chad A. Willis, BS

2d Lt, USAF

Committee Membership:

Dr. Douglas D. Hodson, PhD

Chair

Maj. Richard A. Dill, PhD

Member

Dr. Scott L. Nykl, PhD

Member

AFIT-ENG-MS-21-M-XXX

# Abstract

**Acknowledgments**

Stuff….

Chad A. Willis

**This Note is not to be included with the Acknowledgments – it is for information only: *It is prohibited to include any personal information in the following categories about U.S. citizens, DOD Employees and military personnel: social security account numbers; home addresses; dates of birth; telephone numbers other than duty officers which are appropriately made available to the general public; names, locations and any other identifying information about family members of DOD employees and military personnel.***

# Table of Contents

Page

[Abstract iv](#_Toc48512972)

[Table of Contents vi](#_Toc48512973)

[List of Figures viii](#_Toc48512974)

[List of Tables ix](#_Toc48512975)

[I. Introduction 1](#_Toc48512976)

[General Issue 1](#_Toc48512977)

[Problem Statement 1](#_Toc48512978)

[Research Objectives/Questions/Hypotheses 1](#_Toc48512979)

[Research Focus 1](#_Toc48512980)

[Investigative Questions 1](#_Toc48512981)

[Methodology 1](#_Toc48512982)

[Assumptions/Limitations 1](#_Toc48512983)

[Implications 2](#_Toc48512984)

[Preview 2](#_Toc48512985)

[II. Background 2](#_Toc48512986)

[2.1 Chapter Overview 2](#_Toc48512987)

[2.2 Rust 2](#_Toc48512988)

[2.3 Rust and Networking 3](#_Toc48512989)

[2.4 Entity Component System 3](#_Toc48512990)

[2.5 Physics of Flight Modeling 4](#_Toc48512991)

[2.6 FlightGear 4](#_Toc48512992)

[III. Methodology 4](#_Toc48512993)

[Chapter Overview 4](#_Toc48512994)

[Test Subjects 4](#_Toc48512995)

[Summary 4](#_Toc48512996)

[IV. Analysis and Results 5](#_Toc48512997)

[Chapter Overview 5](#_Toc48512998)

[Results of Simulation Scenarios 5](#_Toc48512999)

[Investigative Questions Answered 5](#_Toc48513000)

[Summary 5](#_Toc48513001)

[V. Conclusions and Recommendations 5](#_Toc48513002)

[Chapter Overview 5](#_Toc48513003)

[Conclusions of Research 6](#_Toc48513004)

[Significance of Research 6](#_Toc48513005)

[Recommendations for Action 6](#_Toc48513006)

[Recommendations for Future Research 6](#_Toc48513007)

[Summary 6](#_Toc48513008)

[Appendix 7](#_Toc48513009)

[Bibliography 8](#_Toc48513010)

[Vita 9](#_Toc48513011)

# List of Figures

Page

[Figure 1. Figure Example 3](#_Toc233011961)

# List of Tables

Page

[Table 1. Table Example 3](#_Toc233011979)

A RUST BASED, ECS PATTERNED, INTERACTIVE FLIGHT SIMULATOR

# I. Introduction

## General Issue

Text

Text

“Quoted text.”

## Problem Statement

Text

## Research Objectives/Questions/Hypotheses

Text

## Research Focus

Text

## Investigative Questions

Text

## Methodology

Text

## Assumptions/Limitations

Text

## Implications

Text

## Preview

Text

# II. Background

## 2.1 Chapter Overview

This chapter will give an overview of the necessary information to follow this thesis. This overview will provide the fundamentals of the Rust programming language. With that, basic networking, specifically with Rust, will be explained. Next, the Entity Component System (ECS) architecture, and an implementation of it, Specs, will be described. Furthermore, the physics for flight dynamics modeling will be laid out. Finally, the use of the FlightGear flight simulator will be explained.

## 2.2 Rust

Rust is the programming language being leveraged to build a flight dynamics model. It is the language of choice given its memory safety and speed. It was released in 2015 by Mozilla, which has been behind the success of Rust [5]. Rust can guarantee that a program will be memory safe, with no invalid data accesses, all while remaining just as fast as other popular programming languages. The C++ programming languages gives developers complete access to data inside memory, although it gives no safety precautions. The python programming language gives developers complete memory safety, but developers lose control of data inside memory [5]. Rust combines the strengths of both C++, and Python. It guarantees memory safety and allows complete access to that memory – with exceptional performance.  
 The safety assurance of Rust is accomplished by its system of ownership, borrowing, and lifetimes. All languages have a way to manage memory while a program is running. Some languages, like java, have a garbage collector, which actively checks for memory that is no longer being used during runtime. Other languages, like C++, make the programmer manually allocate and then free memory. Rust does not take the approach of either Java or C++; it instead has something called a “borrow checker.” At compile time, the Rust borrow checker will check that all accesses to data is legal and valid [5]. The borrow checker deals with three important Rust concepts: ownership, borrowing, and lifetimes.

Ownership refers to the fact that every use of a value needs to be valid and cleaned up when it is done being used. Ownership is accomplished with three rules that the programmer must follow in Rust [3]:

1. Every value has a variable which is called its owner,
2. There may be only one owner at a time
3. Values are dropped when an owner goes out of scope.

**fn** main**()**

**{**

//Example 1

**{**

//the value 24 has a sole owner, x

**let** x **=** 24**;**

//valid

println!**(**"x: {}"**,** x**);**

**}**

// Error: x is no longer in scope, so the value is dropped

println!**(**"x: {}"**,** x**);**

//Example 2

//initialize a, which is an owner of a vector

**let** a **=** vec!**[**1**,** 2**,** 3**];**

//reassignment of ownership, b now owns the value which a previously owned. a becomes uninitialized and not usable

**let** b **=** a**;**

// Error: moved value a is trying to be accessed

println!**(**"a: {} , b: {}"**,** a**,** b**);**

**}**

1 <https://depth-first.com/articles/2020/01/27/rust-ownership-by-example/>

2 <https://doc.rust-lang.org/nomicon/ownership.html>

3 <https://doc.rust-lang.org/1.9.0/book/ownership.html>

4 <https://blog.gds-gov.tech/appreciating-rust-memory-safety-438301fee097>  
5 Rust in action book

Borrowing is the next important concept in Rust. Borrowing allows data to be accessed for a moment without changing ownership of that data. When ownership of memory is transferred to another binding, the original binding cannot be used, so borrowing is the solution. The borrow checker will statically guarantee that references always point to valid objects. So, if a reference to an object exists, it cannot be destroyed. There are two rules to borrowing:

1. Any borrow must last for a scope no greater than that of the owner,
2. You may have one or the other of these two kinds of borrows, but not both at the same time [7]:
3. One or more references (&T) to a resource
4. exactly one mutable reference (&mut T)

These rules avoid the possibility of a data race because although there can be more than one pointer to a resource, only one can be accessing and changing that resource at a time. This is good for performance, and uses less memory overhead [5].

**fn** main**()** **{**

**let** a **=** vec!**[**1**,** 2**,** 3**];**

//precede the assignee variable with an ampersand to signify a borrow of a value

**let** b **=** **&**a**;**

//this compiles, we can borrow the value of a

println!**(**"a: {} , b: {}"**,** a**,** b**);**

**}**

6 <https://depth-first.com/articles/2020/01/27/rust-ownership-by-example/>

7 <https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html>

The next important Rust concept is lifetimes. A value has a lifetime which is the period that accessing that value is valid to do [8]. A lifetime starts when a value is created and that lifetime ends when it is destroyed. Rust’s compiler is smart enough to know a values lifetime in many common circumstances [6]. In these common circumstances, the lifetime of a value does not need to be explicitly written, but otherwise, a lifetime parameter needs to be written. The borrow checker tries to limit the period of a lifetime because a smaller lifetime would mean that there is less chance of unintended data being referenced, which is called a dangling reference [5]. Consider this example:

//we do not want the function to return a reference to a value which its lifetime ended

**fn** longest**(**x**:** **&str,** y**:** **&str)** **->** **&str** **{** // Error: expects lifetime parameter

**if** x**.**bytes**().**len**()** **>** y**.**bytes**().**len**()** **{**

x

**}** **else** **{**

y

**}**

**}**

//this function defines a lifetime with the angle brackets

//the name of a lifetime parameter begins with an apostrophe (e.g., 'a)

//the lifetime is needed because the compiler needs to know that the functions borrowed return value's lifetime matches that of its x and y parameter

**fn** longest**<**'a**>(**x**:** **&**'a **str,** y**:** **&**'a **str)** **->** **&**'a **str** **{**

**if** x**.**bytes**().**len**()** **>** y**.**bytes**().**len**()** **{**

x

**}** **else** **{**

y

**}**

**}  
[7]**

8 <https://doc.rust-lang.org/book/ch10-03-lifetime-syntax.html>

With safety dealt with by the borrow checker at compile time, the programmer may be at ease when it comes to memory safety. At the same time, there is no cost in performance at runtime because everything is checked at compile time. The downsides are minimal, but they do exist. For one, compile time is longer. Also, the compiler is very strict. New programmers to rust often “fight with the borrow checker” to get their code to compile because it is more difficult to write code that abides by Rusts rules [5]. But the compiler is like a safety net because once the code compiles, the programmer knows it is safe and correct. Less time is spent fixing bugs, and more time is spent getting the compiler right [9].

9 <https://www.youtube.com/watch?v=FYGS2q1bljE> All about rust

Overall, Rust can hold its own against more popular languages, and the learning curve of the language is met with staggering benefits in safety and performance. One will learn that programming in Rust is less about execution flow and what the CPU is doing, and is more about how data is laid out in memory, and how ownership of memory is given to different parts of the program. The Rust community is active and quickly growing. The Rust ecosystem is also expanding rapidly, providing numerous libraries, which Rust calls crates.   
  
Traits…Error Handling? Probably not needed

<https://doc.rust-lang.org/1.9.0/book/>

## 2.3 Rust and Networking

Networking is crucial to the flight simulation created with Rust. Networking is needed to send all of the data from the physics calculations to the visual system so that it knows how what to display on the screen. Networking can simply be defined as two programs communicating with each other, sending and receiving data. To do this, two sockets are required at each end of communication. In this case, the type of socket being used is the Datagram socket. The protocol involving Datagram sockets is called the User Datagram Protocol (UDP). This protocol is considered “connectionless” because an open connection does not need to be maintained. Packets containing bytes of data may simply be built and sent to a destination without a connection needing to be established prior [3].

In Rust, a UdpSocket can be created and binded to an ip address and port on the local machine. After this, data can be sent and received from any other socket address. The socket created can then be connected to the other socket with its ip address and port. From here, data can be sent using the send function, and data can be received using the recv function [10].

**fn** main**()**

**{**

//bind to an ip address

**let** socket **=** UdpSocket**::**bind**(**"127.0.0.1:1337"**).**expect**(**"couldn't bind to address"**);**

//connect to other program on a specified address and port

socket**.**connect**(**"127.0.0.1:5500"**).**expect**(**"connect function failed"**);**

//send the packet of data in the form of bytes

socket**.**send**(&[**0**,** 1**,** 2**,** 4**]).**expect**(**"couldn't send message"**);**

**}**

[11]

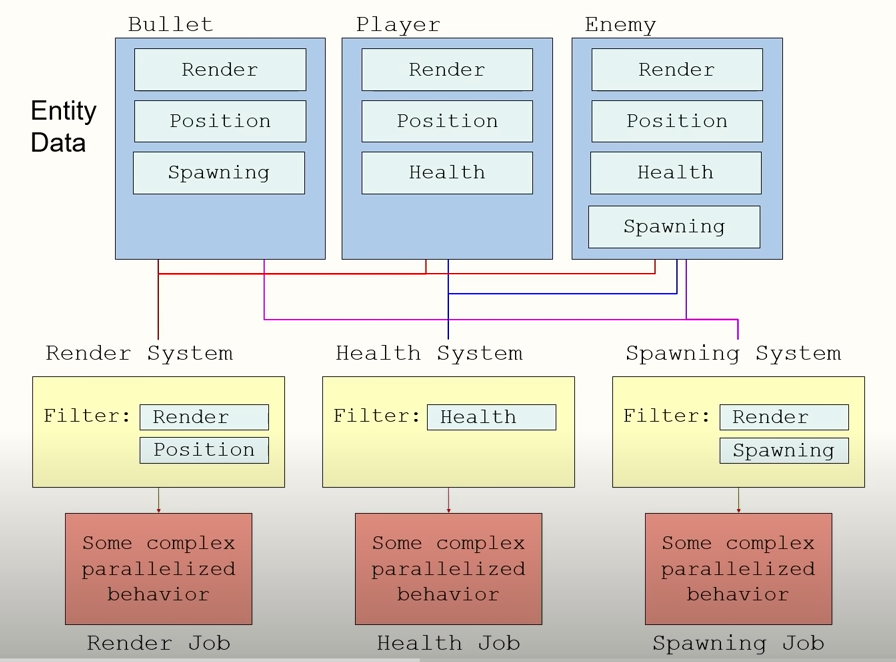
A machines computer architecture represents bytes in a specific order. There are two byte orderings: most significant byte comes first in the ordering, and least significant byte first comes in the ordering. Most significant byte first is called Network Byte order, and least significant byte first is called Host Byte Order. When sending and receiving data, one must be aware of the order on either machine which is necessary to interpret the bytes correctly. So the order may need to be converted first before packets of data are sent from one socket to another [10].

10 Beej’s Guide to Network Programming **-** Using Internet Sockets  
11 <https://doc.rust-lang.org/beta/std/net/struct.UdpSocket.html>

## 2.4 Entity Component System

The ECS architecture, which revolves around these things called Entities, Components, and Systems, is a viable replacement to Object Oriented Programming (OOP). ECS structure is popular in designing game engines, such as Unity. This structure will be used to design our simulation. ECS is essentially a different way to organize data. Where OOP focuses on inheritance and hierarchies, ECS focuses on composition. ECS gets rid of the troubles that arise with inheritance. Suppose we have a space or world that we want to populate with things. These things may have similar attributes as well as unique attributes. With OOP, you would have a base class called Thing that all other Thing types will inherit. The limitation is that one child class can only inherit directly from one parent. For example, a Mammal type may have a child Dog which inherits a fur attribute, and a Bird may have a child Duck which inherits the lays eggs attribute. But in this case, a platypus, which has fur and lays eggs needs to inherit both attributes [15]. This complicates things. The hierarchy chains that are created in game design can become too large to work with and code becomes too highly dependent on other code [12]. This makes maintenance of code and adding new features a pain with OOP. This is where ECS shines because it decouples code.

In an ECS structure, you do not have hierarchy levels. You start with something called an Entity, which represent concrete objects in the world, like a bullet. Entities on their own are useless; they are just containers which Components can be added to. Components hold data and represent a feature. For example, Position is a Component of an Entity. The Components called Position, Render, and Spawning may be added to the Bullet Entity. The mixing and matching of Components can create unique Entities. A Player Entity can be created by adding the Components Position, Render, and Health. Systems are what give functionality to an ECS architecture. A System could be created called Render, which handles all of the rendering functionality of the world. Or a System called Health could handle all of the health functions. Systems have something called a filter. Filters look for every Entity which has some combination of specified Components. For example, the Render System’s filter wants to find every Entity with the Render data Component and Position data Component. Once these Entities are found, they are returned to the Render System to perform some job on it [16]. The System does not care what Entity it is performing the job on. The whole point is that the data Components are completely separated from the Systems functionality to make code easy to refactor and highly parallel [17].



[16]

The ECS structure will be implemented in the simulation created. To implement the ECS in Rust, the Specs will be used. Specs is library, usually used for games and simulations, which makes it easy to build a program leveraging the ECS structure. Specs allows for code that is highly parallel, flexible, and fast [17].   
  
  
12 <http://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-of-mmog-development-part-2/>

13 <https://learn.unity.com/tutorial/entity-component-system#5c7f8528edbc2a002053b67b>  
14 <https://www.youtube.com/watch?v=aKLntZcp27M&t=1775s> Catherine west  
15 v<https://www.youtube.com/watch?v=2rW7ALyHaas> ECS in 7 minutes  
16 <https://www.youtube.com/watch?v=z9WE3fwre-k> ECS unity introduction part 2  
17 <https://specs.amethyst.rs/docs/tutorials/>

## 2.5 Physics of Flight Modeling

## 2.6 FlightGear

FlightGear is an open-source flight simulator application. In the beginning stage of this thesis, FlightGear was leveraged as a visual system. It was used to receive network UDP packets and display the result on the simulation.

# III. Methodology

## Chapter Overview

The purpose of this chapter is

## Test Subjects

Text

## Summary

Text

# IV. Analysis and Results

## Chapter Overview

Text

## Results of Simulation Scenarios

Text

number Equation

Where:

X *bar* = population mean of sorties produced by squadron (LCOM)

o = Actual number of sorties produced (Actual FY2002)

S = Standard deviation of the population mean of sorties produced (LCOM)

n = 75 replications

## Investigative Questions Answered

Text

## Summary

Text

# V. Conclusions and Recommendations

## Chapter Overview

Text

## Conclusions of Research

Text

## Significance of Research

Text

## Recommendations for Action

Text

## Recommendations for Future Research

Text

## Summary

Text

# Appendix

# Bibliography

Abalateo, Mila D. and Lee, Joni R. “Benchmarking Practices of Air Cargo Carriers: A Case Study Approach,” AFIT Thesis AFIT/GLM/LAL/93S-1, (Sep 93)

American National Standards Institute (ANSI). <http://www.ansi.org/> Accessed 15 Dec 2002.

# Vita

OPTIONAL

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **REPORT DOCUMENTATION PAGE** | | | | | | | | | *Form Approved*  *OMB No. 074-0188* |
| The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.  **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.** | | | | | | | | | |
| **1. REPORT DATE** *(DD-MM-YYYY)*  22-03-2012 | | | **2. REPORT TYPE**  Master’s Thesis | | | | | | **3. DATES COVERED** *(From – To)*  March 2010 – March 2012 |
| TITLE AND SUBTITLE  Title of Thesis in Title Case | | | | | | | | **5a. CONTRACT NUMBER** | |
| **5b. GRANT NUMBER** | |
| **5c. PROGRAM ELEMENT NUMBER** | |
| 1. **AUTHOR(S)**   Public, Joseph Q., Captain, USAF | | | | | | | | **5d. PROJECT NUMBER** | |
| **5e. TASK NUMBER** | |
| **5f. WORK UNIT NUMBER** | |
| **7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)**  Air Force Institute of Technology  Graduate School of Engineering and Management (AFIT/ENY)  2950 Hobson Way, Building 640  WPAFB OH 45433-8865 | | | | | | | | | **8. PERFORMING ORGANIZATION**  **REPORT NUMBER**  AFIT-ENY-MS-15-M-000 |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**  AGENCY (spelled out)  ADDRESS  PHONE and EMAIL  ATTN: POC  (no sponsor enter: Intentionally left blank) | | | | | | | | | **10. SPONSOR/MONITOR’S ACRONYM(S)**  **AFRL/RHIQ (example)** |
| **11. SPONSOR/MONITOR’S REPORT NUMBER(S)** |
| **12. DISTRIBUTION/AVAILABILITY STATEMENT**  **DISTRUBTION STATEMENT A.** APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. | | | | | | | | | |
| **13. SUPPLEMENTARY NOTES**   This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. | | | | | | | | | |
| **14. ABSTRACT**  (The abstract can be 200 words maximum.) | | | | | | | | | |
| **15. SUBJECT TERMS**  (Fill in with pertinent terminology related to the topic of your thesis.) | | | | | | | | | |
| **16. SECURITY CLASSIFICATION OF:** | | | | **17. LIMITATION OF**  **ABSTRACT**  **UU** | **18. NUMBER**  **OF PAGES**  19 | **19a. NAME OF RESPONSIBLE PERSON**  Name of Advisor, AFIT/ENY | | | |
| **a. REPORT**  **U** | **b. ABSTRACT**  **U** | **c. THIS PAGE**  **U** | | **19b. TELEPHONE NUMBER** *(Include area code)*  (937) 255-6565, ext xxxx (NOT DSN)  (emailname@afit.edu) | | | |
|  | | | |  |  | | **Standard Form 298 (Rev. 8-98)**  Prescribed by ANSI Std. Z39-18 | | |